Discrete Applied Mathematics 35 (1992) 237-253
North-Holland

Exchanges procedures for timetabling problems*

Jacques A. Ferland and Alain Lavoie

Laboratoire d'Optimisation, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué., Canada H3C 3J7

Received 18 January 1990
Revised 1 September 1990

Abstract

Ferland, J.A. and A. Lavoie, Exchanges procedures for timetabling problems, Discrete Applied Mathematics 35 (1992) 237-253.

Timetabling problems appear in several practical applications, and they can be formulated as 0-1 programming problems to determine an optimal assignment of items to resources minimizing total cost and satisfying K additional side constraints. The proposed approach to deal with this problem is a heuristic iterative procedure where the assignment of one item is modified at each iteration. This exchange procedure is applied first to determine a feasible solution satisfying the side constraints, and then to improve the objective function. A geometric interpretation of an exchange is first given to induce a theoretical framework for the procedure. Furthermore, two other procedures are introduced to prevent jamming situations outside the feasible domain or at a local optimum. The first procedure uses inductively more than one exchange per iteration, and the second one relies on Lagrangean relaxation.

1. Introduction

Timetabling problems appear in several practical applications: courses timetabling [1,2,6,20], exams timetabling [3,16], sport leagues games scheduling [5,9], etc. A formal definition of a typical timetabling problem is as follows:

> Given n items and m resources, denote by c_{ij} the cost of assigning (scheduling) item i to resource j. The problem is to determine an optimal assignment of items to resources minimizing total cost and satisfying K additional side constraints.

* This research was supported by an NSERC (Grant A8312).

A mathematical model for this problem can be formulated as follows:

$$\text{(TP)} \quad \text{Min} \quad f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij},$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} = 1, \qquad 1 \le i \le n, \tag{1.1}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ijk} x_{ij} \le b_k, \qquad 1 \le k \le K, \tag{1.2}$$

$$x_{ij} = 0 \text{ or } 1, \qquad 1 \le i \le n, \quad 1 \le j \le m, \tag{1.3}$$

where x_{ij} is a decision variable

$$x_{ij} = \begin{cases} 1, & \text{if item } i \text{ is assigned to resource } j, \\ 0, & \text{otherwise.} \end{cases}$$

The items assignment constraints are formulated in (1.1) and the side constraints in (1.2). Note that the timetabling problem (TP) is sometimes referred to as an assignment type problem with side constraints.

The side constraints are useful to specify appropriate timetables in different contexts. In [1,2], this formulation is used for the courses timetabling problem. The variable x_{ij} is equal to 1 whenever lecture i is starting at period j. Some side constraints are used to eliminate conflicts due to students required to take or instructors required to teach simultaneously more than one lecture. Other side constraints are used to eliminate shortage of classrooms in specific periods. The objective function is specified in terms the instructors' availabilities and preferences.

In [5,9], (TP) is used to formulate a sport league scheduling problem. The variable x_{ij} is equal to 1 whenever game i is played on day j. The side constraints are used to specify the availability of the arenas, some restrictions on the number of games on consecutive days for a team, restrictions on minimum time between revisits of one team to another team, and other restrictions specific to the league. The objective is in general to reduce the total distance travelled by each team.

A well-known special case of (TP) is the generalized assignment problem (GAP) [8,19] where the side constraints (1.2) correspond to the capacity constraints of the resources as follows:

$$\sum_{i=1}^{n} r_{ij} x_{ij} \le b_j, \quad 1 \le j \le m. \tag{1.4}$$

This problem is known to be NP-complete (see [18]). The 0-1 assignment problem with side constraints (APSC), analyzed by Mazzola and Neebe in [18], can also be seen as a special instance of (TP) where the side constraints are partitioned into two groups:

(i) the resources assignments constraints

$$\sum_{i=1}^{n} x_{ij} = 1, \quad 1 \le j \le m, \tag{1.5}$$

(ii) the K' ($K'=K-m$) side constraints

$$\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ijk}x_{ij}\leq b_k, \quad 1\leq k\leq K'.$$

In practical applications, the number of items (n) and the number of side constraints (K) grow rapidly. Hence, we have to deal with large scale 0-1 mathematical programming problems. Furthermore, the NP-completeness of the (GAP) urges us to use efficient heuristic procedures. Moreover, practical applications are always complex, and several hidden constraints may not be included in the model because of the difficulty in formulating them mathematically. Hence, these procedures have to be embedded in user friendly decision support systems [4,5].

The proposed approach to deal with (TP) is a heuristic iterative method where the iterate points satisfy constraints (1.1) and (1.3). It includes two phases: in Phase I, we are searching for a feasible solution (i.e., a point satisfying also (1.2)), and in Phase II, we improve the value of the objective function $f(x)$. At each iteration during either phase, an exchange is performed; i.e., the assignment of an item is modified from one resource to another in order to preserve the feasibility of constraints (1.1) and (1.3). Furthermore, an exchange is always performed to improve a function OPT(x) which differs for the distinct phases: OPT is a function measuring the "distance" to the feasible domain, and in Phase II, OPT(x)=$f(x)$. It is interesting to note the correspondence between OPT and function OPTIMUM introduced by Glover in [13].

The proposed approach was used successfully to deal with the preceding applications (see [1,2,5,6,9,20]). The purpose of this paper is to introduce a more formal presentation of the approach to foresee its applications in other contexts.

In Section 2, a geometric interpretation of an exchange is presented. This interpretation induces a theoretical framework for exchange procedures. The approach is then analytically specified in Section 3. In Sections 4 and 5, two other procedures are introduced to improve the basic exchange procedure and to prevent situations where it fails to reach the feasible domain or where it reaches a local optimum. The first procedure uses inductively several exchanges per iteration and the second relies on Lagrangean relaxation to move away from the jamming point. Finally, numerical results are given in Section 6 to analyze the behavior of these two procedures.

2. Geometric interpretation of an exchange

First, we introduce some basic results for an integer programming problem with structures more general than (TP). For (TP), these results induce a useful geometric interpretation of an exchange as a simplex pivot or a motion from one extreme point of a polyhedral convex set to an adjacent one. Furthermore, they allow the characterization of Mazzola and Neebe approach within this framework.

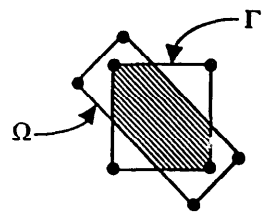

Fig. 1.

2.1. Basic results

The first results are introduced in terms of general polyhedral convex sets. We consider two polyhedral convex sets Γ and Ω in R^n, and we assume that all integer points of Γ are extreme points. For instance, the unit hypercube is such a set. The first result, illustrated in Fig. 1, is a straightforward consequence of the definition of Γ.

Proposition 2.1 [17, Theorem 1.4]. *Let Γ and Ω be polyhedral convex sets in R^n. Assume that all integer points of Γ are extreme points. If $x \in \Gamma \cap \Omega$ is integer, then x is an extreme point of $\Gamma \cap \Omega$.*

Additional assumptions are required for the converse to be true. Such assumptions can be derived by relying on total unimodularity.

Proposition 2.2 [17, Theorem 1.5]. *Let $\Gamma \in R^n$ be the unit hypercube; i.e., $\Gamma = \{x \in R^n: 0 \le x \le 1\}$. Assume that $\Omega = \{x \in R^n: Ax \le b\}$ where A is an $m \times n$ totally unimodular matrix and $b \in R^m$ is a vector of integer values. Then $x \in \Gamma \cap \Omega$ is integer if and only if x is an extreme point of $\Gamma \cap \Omega$.*

A straightforward geometric interpretation of an exchange can now be derived.

2.2. Exchanges to deal with (TP)

Consider the linear programming relaxation $\overline{(\mathrm{TP})}$ of (TP) where (1.3) are replaced by

$$0 \le x_{ij} \le 1, \quad 1 \le i \le n, \ 1 \le j \le m. \tag{2.1}$$

Hence, if we denote

$$\Gamma(\mathrm{TP}) = \{x \in R^{m \times n}: 0 \le x_{ij} \le 1, \ 1 \le i \le n, \ 1 \le j \le m\},$$

$$\Omega(\mathrm{TP}) = \{x \in R^{m \times n}: \sum_{j=1}^{m} x_{ij} = 1, \ 1 \le i \le n\},$$

then Proposition 2.2 (mutatis mutandis where inequalities are replaced by equalities) applies to establish a one to one correspondence between the extreme point of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ and the integer points of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ which are the iterate points of the exchange procedure.

To show that an exchange corresponds to moving from an extreme point of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ to an adjacent one, we use the following equivalence

$$\Gamma(\text{TP}) \cap \Omega(\text{TP}) = X$$

where $X = \{x \in R^{m \times n}: \sum_{j=1}^{m} x_{ij} = 1,\ 1 \le i \le n;\ x_{ij} \ge 0,\ 1 \le i \le n,\ 1 \le j \le m\}$ (i.e., the constraints $x_{ij} \le 1,\ 1 \le i \le n,\ 1 \le j \le m$ are redundant). Now, referring to X, it is easy to see that an exchange corresponds to a simplex pivot in X. Hence, the result is verified.

In Fig. 2, $F(\overline{\text{TP}})$ denotes the feasible domain of $(\overline{\text{TP}})$ and $F((1.2))$ the set of points satisfying constraints (1.2). The feasible domain of (TP), $F(\text{TP})$ reduces to the extreme points of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ in the set $F((1.2))$. In terms of this illustration, the approach (introduced in Section 1) to deal with (TP) is to move between adjacent extreme points of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ to get first into $F((1.2))$. Then the process is repeated to improve $f(x)$ but staying inside $F((1.2))$.

2.3. Exchanges to deal with (APSC)

In [13], Mazzola and Neebe also use an exchange procedure to deal with problem (APSC). In their procedure, the iterate points are the integer points of $\Gamma(\text{APSC}) \cap \Omega(\text{APSC})$ where

$$\Gamma(\text{APSC}) = \{x \in R^{n \times n}: 0 \le x_{ij} \le 1,\ 1 \le i \le n,\ 1 \le j \le n\},$$

$$\Omega(\text{APSC}) = \left\{x \in R^{n \times n}: \sum_{j=1}^{n} x_{ij} = 1,\ 1 \le i \le n;\ \sum_{i=1}^{m} x_{ij} = 1,\ 1 \le j \le n\right\}.$$

Hence, Proposition 2.2 (mutatis mutandis where inequalities are replaced by equalities) also applies. Furthermore, it is also easy to verify that an exchange in

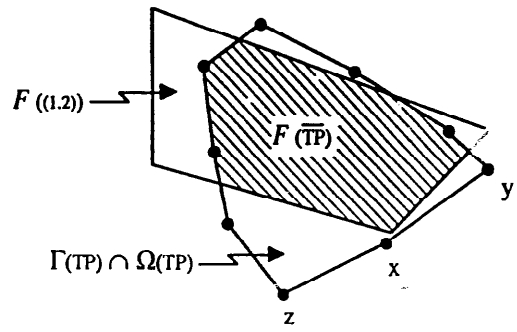

Fig. 2.

their procedure corresponds to moving from an extreme point of $\Gamma(\text{APSC}) \cap \Omega(\text{APSC})$ to an adjacent one.

3. Exchange procedure

To specify the exchange procedure, we use a "distance" function from the feasible domain introduced by Mazzola and Neebe in [18]. It is interesting to note that other definitions could also be used for the distance.

Let x be an extreme point of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$, and define

$$s_k(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ijk} x_{ij} - b_k, \quad 1 \le k \le K.$$

The distance function denoted by $I(x)$ is as follows

$$I(x) = \sum_{k=1}^{K} \text{Max}\{s_k(x), 0\}.$$

Hence, x is feasible for (TP) if $I(x) = 0$.

For each extreme point x of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ denote by $j(i)$ the index $1 \le j \le m$ such that $x_{ij(i)} = 1$ (i.e., item i is assigned to resource $j(i)$). Hence, for all $1 \le i \le n$

$$x_{ij} = \begin{cases} 1, & \text{if } j = j(i), \\ 0, & \text{otherwise.} \end{cases}$$

For $1 \le i \le n$, $1 \le j \le m$, $1 \le k \le K$, denote

$$\bar{r}_{ijk}(x) = r_{ijk} - r_{ij(i)k}.$$

$$\Delta_{ijk}(x) = \begin{cases} \bar{r}_{ijk}(x), & \text{if } s_k(x) \ge 0 \text{ and } \bar{r}_{ijk}(x) > 0, \\ \text{Max}\{\bar{r}_{ijk}(x) + s_k(x), 0\}, & \text{if } s_k(x) < 0 \text{ and } \bar{r}_{ijk}(x) > 0, \\ -\text{Min}\{s_k(x), -\bar{r}_{ijk}(x)\}, & \text{if } s_k(x) \ge 0 \text{ and } \bar{r}_{ijk}(x) < 0, \\ 0, & \text{otherwise.} \end{cases}$$

Hence, $\Delta_{ijk}(x)$ measures some influence on the kth side constraint feasibility if the assignment of item i is modified from $j(i)$ to j. Then

$$\Delta_{ij}(x) = \sum_{k=1}^{K} \Delta_{ijk}(x)$$

measures the influence of this modification for all the side constraints. Furthermore, referring to Proposition 2.2, $\Delta_{ij}(x)$ measures the influence on the distance function $I(x)$ when moving from an extreme point of $\Gamma(\text{TP}) \cap \Omega(\text{TP})$ to an adjacent one. Note that for more general polyhedral convex set Ω, the derivation of Δ might not be as easy as it is for (TP).

The *exchange procedure* can be summarized as follows:

Step 0. Let x be an extreme point of $\Gamma(\mathrm{TP}) \cap \Omega(\mathrm{TP})$.

Step 1. If $I(x)=0$, go to Step 4.

Step 2. For $1 \le i \le n$, $1 \le j \le m$, determine $\Delta_{ij}(x)$.
Determine also

$$\Delta_{pq}(x) = \min_{\substack{1 \le i \le n \\ 1 \le j \le m}} \{\Delta_{ij}(x)\}.$$

If $\Delta_{pq}(x) \ge 0$, then STOP because the procedure fails to identify a feasible solution of (TP).

Step 3. Complete the exchange:

$$x_{pj(p)} = 0 \quad \text{and} \quad x_{pq} = 1.$$

Return to Step 1.

Step 4. (We are in the feasible domain of (TP).)
For $1 \le i \le n$, $1 \le j \le m$, determine $\bar{c}_{ij}(x) = c_{ij} - c_{ij(i)}$.
Determine also:

$$\bar{c}_{pq}(x) = \min_{\substack{1 \le i \le n \\ 1 \le j \le m}} \{\bar{c}_{ij}(x) : \Delta_{ijk}(x) \le 0,\ 1 \le k \le K\}. \tag{3.1}$$

If $\bar{c}_{pq}(x) \ge 0$, then STOP because the procedure cannot find any better solution than x.
Otherwise, go to Step 3.

It is interesting to note that (3.1) reduces to

$$\bar{c}_{pq}(x) = \min_{\substack{1 \le i \le n \\ 1 \le j \le m}} \{\bar{c}_{ij}(x) : \Delta_{ijk}(x) = 0,\ 1 \le k \le K\}.$$

Indeed, it is easy to verify that this follows from the condition $I(x)=0$.

There are two main drawbacks in this simple procedure. First when stopping in Step 2, it fails to identify a feasible solution. Second, when stopping in Step 4, the identified solution may not be optimal. Hence, more powerful procedures are required to prevent these jamming situations. Two different procedures for such situations are now introduced.

4. Recursive procedure

In the preceding procedure, on one hand, an improvement is required ($\Delta_{pq} < 0$ or $\bar{c}_{pq} < 0$) to complete an exchange. On the other hand, the current solution at the outcome of the procedure may be a local optimum of OPT(x) (i.e., $\Delta_{pq}(x) \ge 0$ or $\bar{c}_{pq}(x) \ge 0$) and, to accomplish an improvement, we may have to reach an extreme point which is not adjacent to the current one. Hence, the value of function OPT may have to increase first when reaching adjacent extreme points before decreasing

any further. The recursive procedure to be introduced is based on this observation.

To illustrate the procedure, we refer to Fig. 2. Assuming that x is the solution on hand, then the preceding *exchange procedure* stops because the feasibility measure increases whenever we move to y or z. But, moving to y and reapplying the procedure, the feasibility domain is reached at the next iteration. Hence, the feasibility measure increases first, but it reaches a lower value after two exchanges.

4.1. The procedure

At each iteration of the recursive procedure, several exchanges may be required to improve the function OPT. In our notation, a "*forward motion*" corresponds to an exchange which is performed to move from an extreme point to an adjacent one that has not yet been visited during the iteration. Similarly, a "*backward motion*" corresponds to a move from the current point back to an adjacent extreme point already visited so far during the iteration. Three different features are specified to efficiently manage these motions during an iteration.

First, an upper bound τ is specified to limit the number of forward motions during an iteration. Denote by $c\tau$ the number of forward motions performed so far during the iteration to reach the current extreme point. Hence, $c\tau = 0$ at the outset of the iteration.

The second feature is characterized by an upper bound MR on the net total increase of the function OPT admissible in the forward motions to reach an alternative extreme point from x^0, the current extreme point at the outset of the iteration. Denote by R the net total increase of the forward motions performed so far to reach the current point x from x^0. Hence, $R = 0$ at the outset of the iteration, and if the forward motion to leave the current point x is specified by the pair (p, q), then R is updated as follows:

$$R = R + \Delta_{pq}(x) \quad \text{or} \quad R = R + \bar{c}_{pq}(x).$$

Now, since a positive net total increase is allowed when forward motions are performed, an additional feature is required to prevent cycling through a subset of extreme points. For this purpose, denote by T the list of extreme points visited so far during the iteration in forward motions to reach the current point. Hence, at the outset of the iteration, $T = \{x^0\}$. Furthermore, the current point reached in the forward motion is included in the list T. Also with each extreme point x visited during the iteration, associate a list $T(x)$ of extreme points adjacent to x and reached from x during the iteration. Hence, whenever x is reached, then $T(x)$ is initiated as an empty list, and $T(x^0) = \emptyset$ at the outset of the iteration.

Finally, the set $S(x)$ includes three subsets of pairs (i,j) ranked in increasing lexicographic order: first the subset of pairs (i,j) with $\Delta_{ij}(x) < 0$ ($\bar{c}_{ij}(x) < 0$), then the subset of those with $\Delta_{ij}(x) = 0$ ($\bar{c}_{ij}(x) = 0$), and finally the subset of those with $\Delta_{ij}(x) > 0$ ($\bar{c}_{ij}(x) > 0$). To determine the forward motion to leave the current extreme point x we select the first pair $(i,j) \in S(x)$ characterizing an exchange which

is not leading to any adjacent point of x already in the list T. This selection strategy requires less computing effort than the one relying on the criteria in Step 2 or Step 4 of the *exchange procedure* in Section 3 where the minimum is taken over all pairs (i,j) that are not leading to any adjacent point in the list T.

To describe an iteration of the recursive procedure, assume that x is the current extreme point, that $c\tau$ ($\leq \tau$) is the number of forward motions to reach x from x^0, and that R is the net total increase. Furthermore, denote by z the extreme point from which x has been reached and $(\bar{p},\bar{q})$ the pair characterizing the corresponding forward motion. Several cases have to be analyzed:

(i) If $R<0$, then the iteration is completed and the current extreme point x becomes the initial point for the next iteration.

(ii) If $R\geq 0$, and $c\tau=\tau$ or $R>\mathrm{MR}$, then a backward motion is performed to return to z. Hence,

$$c\tau = c\tau - 1,$$

$$R = R - \Delta_{\bar{p}\bar{q}}(z)\ (R - c_{\bar{p}\bar{q}}(z)),$$

z is the new current point.

(iii) If $R\geq 0$, $c\tau<\tau$, $R\leq \mathrm{MR}$, and an exchange (p,q) can be identified by the above procedure to characterize a forward motion from x to an adjacent point y, then the forward motion is performed to reach y and

$$c\tau = c\tau + 1,$$

$$R = R + \Delta_{pq}(x)\ (R + c_{pq}(x)),$$

$$T(x) = T(x) \cup \{y\},$$

$$T = T \cup \{y\},$$

$$T(y) = \emptyset,$$

y is the new current point.

(iv) If $R\geq 0$, $c\tau<\tau$, $R\leq \mathrm{MR}$, but an exchange cannot be identified by the above procedure (i.e., all adjacent extreme points of x are in the list T), then a backward motion is performed to return to z. Hence,

$$c\tau = c\tau - 1,$$

$$R = R - \Delta_{\bar{p}\bar{q}}(z)\ (R - c_{\bar{p}\bar{q}}(z)),$$

$$T = T - T(x),$$

z is the new current point.

Note that, in this case, if $x=x^0$, then a backward motion cannot be performed, and the recursive procedure stops because it cannot identify another extreme point where the value of the function OPT is smaller for these values of the parameters τ and MR.

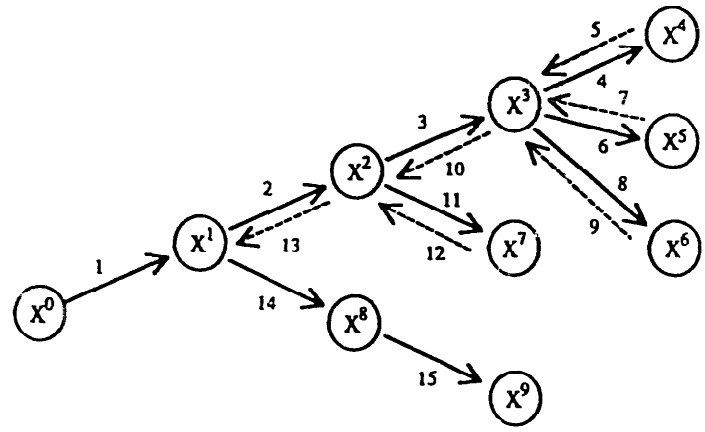

Fig. 3.

Figure 3 illustrates an iteration where $\tau = 4$.

The index i of node x^i indicates that it is the ith point visited. The arc index j indicates that this arc corresponds to the jth motion of the iteration. Furthermore, Fig. 4 gives the elements in T after each motion. In particular, we note that

Motion	Element in list T after motion
1	x^0, x^1
2	x^0, x^1, x^2
3	x^0, x^1, x^2, x^3
4	x^0, x^1, x^2, x^3, x^4
5	x^0, x^1, x^2, x^3, x^4
6	$x^0, x^1, x^2, x^3, x^4, x^5$
7	$x^0, x^1, x^2, x^3, x^4, x^5$
8	$x^0, x^1, x^2, x^3, x^4, x^5, x^6$
9	$x^0, x^1, x^2, x^3, x^4, x^5, x^6$
10	x^0, x^1, x^2, x^3
11	x^0, x^1, x^2, x^3, x^7
12	x^0, x^1, x^2, x^3, x^7
13	x^0, x^1, x^2
14	x^0, x^1, x^2, x^8
15	x^0, x^1, x^2, x^8, x^9

Fig. 4.

whenever x^4, x^5, x^6 are reached, the value of R is positive and backward motions are required to return to x^3. Also, when backward motion 10 is performed to return to node x^2, then x^4, x^5 and x^6 are eliminated from T but x^3 remains in T (see case (iv)). The purpose is to allow the possibility of returning to x^4, x^5, x^6 using another path, but to eliminate any cycle with x^3 in it. Finally, in this example, the iteration is completed after 15 moves to reach node x^9 where the function takes a smaller value than at x^0.

Like the *exchange procedure*, the *recursive procedure RP* includes two major phases. During Phase I, the procedure is trying to reach the feasible domain by improving the feasibility measure (corresponding to Steps 2 and 3 of the exchange procedure). Once the feasible domain is reached (if ever), then the procedure improves the objective function in Phase II (corresponding to Steps 3 and 4 of the exchange procedure). In general, the parameters τ and MR are different during the different phases.

It is interesting to note that if $\tau = 1$ in both phases of the recursive procedure, then we obtain a variant of the exchange procedure where the exchange selection (see above) is different from the criteria in Steps 2 and 4.

4.2. Relations with tabu search approach

The preceding *recursive procedure RP* has some similarities with the *tabu search* approach [11, 13, 15]. Indeed, in both approaches the current solution is obtained by moving from an extreme point of a polyhedral convex set to an adjacent one. The list T in the recursive procedure can be seen as some kind of "tabu list" with a particular updating mechanism. Finally, the *backward motion* is related to the so called "*aspiration function*", to offset the tabu status of an extreme point already visited with the aspiration of starting in a new direction inducing better improvement. In contrast with the tabu search approach, the backward motion is systematic and is not conditional on the induced modification on the value of R (i.e., on the value of the function).

5. Lagrangean relaxation

In the *recursive procedure RP*, whenever several exchanges are performed during an iteration, it means that we are moving to an extreme point which is not adjacent to a current local optimum. Such an extreme point can be identified using an alternate procedure based on Lagrangean relaxation. This approach has been used by Mazzola and Neebe [18] to deal with problem (APSC) where Γ(APSC) and Ω(APSC) are specified in Section 2.3.

Consider the Lagrangean relaxation of (TP) specified with the multipliers λ_k $(1 \leq k \leq K)$ associated with the side constraints:

$$(\mathrm{TPR}_\lambda) \quad \text{Min} \quad \left\{ \sum_{i=1}^{n} \sum_{j=1}^{m} \left[c_{ij} + \sum_{k=1}^{K} \lambda_k r_{ijk} \right] x_{ij} \right\} - \sum_{k=1}^{K} \lambda_k b_k,$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} = 1, \quad 1 \le i \le n,$$

$$x_{ij} = 0 \text{ or } 1, \quad 1 \le i \le n, \ 1 \le j \le m.$$

It is interesting to note that (TPR_λ) is trivial to solve. Furthermore, it is well known [10] that for any vector $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_k] \ge 0$, the optimal value $v(\mathrm{TPR}_\lambda)$ of (TPR_λ) is a lower bound on the optimal value $v(\mathrm{TP})$ of (TP); i.e., for all $\lambda \ge 0$

$$v(\mathrm{TPR}_\lambda) \le v(\mathrm{TP}).$$

Hence, to identify a better lower bound, we may solve the Lagrangean dual of (TP); i.e., solve

$$(\mathrm{TPD}) \quad \max_{\lambda \ge 0} v(\mathrm{TPR}_\lambda).$$

The *procedure LR* to deal with (TP) corresponds to solving (TPD). It is summarized as follows:

Step 0. Let $\bar{\lambda} \ge 0$. Determine x an optimal solution of $(\mathrm{TPR}_{\bar{\lambda}})$. Let

$$\mathrm{LB} = v(\mathrm{TPR}_{\bar{\lambda}}),$$

$$\mathrm{UB} = \sum_{i=1}^{n} \left\lceil \max_{1 \le j \le m} \{c_{ij}\} \right\rceil,$$

$$x^* = 0 \in R^{n \times m}.$$

Step 1. Apply the *exchange procedure* (or *RP* with $\tau = 1$ in both phases), with the initial extreme point x. If the exchange procedure is successful in identifying a feasible solution of (TP), let $\tilde{x}$ denote this solution. If $\mathrm{UB} \ge \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} \tilde{x}_{ij}$, then $\mathrm{UB} = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} \tilde{x}_{ij}$, and $x^* = \tilde{x}$.

Step 2. Given the current feasible solution $\bar{\lambda}$ of (TPD), execute one iteration of a procedure to deal with (TPD), and determine the next iterate $\tilde{\lambda}$. Then let $\bar{\lambda} \leftarrow \tilde{\lambda}$. Determine an optimal solution x of $(\mathrm{TPR}_{\bar{\lambda}})$. Let

$$\mathrm{LB} = \mathrm{Max}\{\mathrm{LB}, v(\mathrm{TPR}_{\bar{\lambda}})\}.$$

Step 3. If $(\mathrm{UB} - \mathrm{LB})/\mathrm{LB} > \varrho$, return to Step 1.
Otherwise, STOP because x^* is an ε-optimal solution of (TP) where $\varepsilon = \mathrm{UB} - \mathrm{LB}$.

Since $v(\mathrm{TPR}_\lambda)$ is not in general differentiable, we use a subgradient procedure [7, 14] to deal with (TPD) in Step 2. In our experimentation $\tilde{\lambda}$ is determined according to the procedure proposed by Held, Wolfe and Crowder [14], i.e.

$$\tilde{\lambda}_k = \text{Max}\left\{0, \bar{\lambda}_k + \frac{\tau(\bar{\lambda})s_k(x)[\text{UB} - v(\text{TPR}_{\bar{\lambda}})]}{\sum_{k=1}^{K} s_k^2(x)}\right\}$$

where $0 < \tau(\bar{\lambda}) < 2$. The parameter ϱ in Step 3 has to be specified appropriately to account for the duality gap $v(\text{TP}) - v(\text{TPD})$.

6. Numerical results

Numerical results are now used to compare the recursive *procedure RP* and the variant of the Lagrangean relaxation *procedure LR* where Step 1 is completed using *RP* with $\tau = 1$ in both phases. The procedures are implemented in Turbo Pascal (version 4.0), and the problems are solved on an IBM/PS-2-50z. Most of these problems are also solved using XMP, a general integer programming code on a SUN-4 workstation. However, in all the following tables, the SUN-4 CPU time is transformed into IBM/PS-2-50z CPU time according to their relative performance (10 MIPS for SUN-4 workstation and 1.4 MIPS for IBM/PS-2-50z).

The problems are generated according to a process proposed by Mazzola and Neebe [18]. The cost coefficients c_{ij} and the coefficients r_{ijk} in the additional side constraints are selected according to a uniform distribution over the interval [1, 250]. Furthermore, the vector of the cost coefficients and the matrix corresponding to the additional side constraints coefficients are 100% dense. Finally, the right-hand side values of the constraints are initially specified as follows:

$$b_k = 0.8 \left\lfloor \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ijk}}{n} \right\rfloor, \quad 1 \le k \le K$$

where $\lfloor a \rfloor$ denotes the largest integer smaller or equal to a.

The value of parameter τ is fixed at 10 during Phase I and 3 during Phase II.

In *procedure LR*, $\tau(\bar{\lambda}) = 0.025$, $\varrho = 0.02$, and the number of iterations is limited to 20. Time limits are specified as follows: 20 minutes for *RP* and *LR* and 7 hours for XMP.

The initial solution for *RP* is specified as follows

$$j(i) = \text{Min}\left\{j: c_{ij} = \underset{1 \le j \le n}{\text{Min}} \{c_{ij}\}\right\}.$$

The initial value of $\bar{\lambda}$ in LR is equal to 0. Hence, in fact, both procedures start from the same extreme point.

For both procedures, a relative deviation RD of the current best solution (having value UB) from the optimal value is computed as follows:

$$\text{RD} = \frac{\text{UB} - \text{LB}}{\text{LB}}.$$

In Tables 1, 2 and 4, LB is either the optimal value or the best lower bound obtained

Table 1. (TP) problems with one additional side constraint

			RP							LR		XMP
			Phase I			Phase II						
n	m	k	$c\tau_1$	MR	CPU (sec)	$c\tau_2$	MR	total CPU (sec)	RD	RD	CPU (sec)	CPU (sec)
10	10	1	1	0	1	1	20	3	0.56	0	12	14
20	20	1	1	0	5	3	20	19	0.06	0	103	58
30	30	1	1	0	14	2	20	36	0.18	0	101	115
40	40	1	1	0	55	2	20	93	0.32	0	1206	367
50	50	1	1	0	31	1	20	67	0.11	0	1210	504
60	60	1	1	0	127	3	20	429	0.10	0	1215	6173
70	70	1	1	0	185	2	20	395	0.09	0	1219	8345
80	80	1	1	0	444	3	20	1100	0.03	0	1234	3640

with XMP. In Table 3, RD is computed using the value of LB obtained by solving the problem with *LR*.

Finally, in the tables of results, we indicate the largest number of forward motions used in any iteration for each phase of the *RP* procedure. $c\tau_1$ and $c\tau_2$ denote these numbers for Phase I and II, respectively.

Table 2. (TP) problems with several additional side constraints (* indicates that RP, LR and XMP fail to identify a feasible solution)

			RP							LR		XMP
			Phase I			Phase II						
n	m	k	$c\tau_1$	MR	CPU (sec)	$c\tau_2$	MR	total CPU (sec)	RD	RD	CPU (sec)	CPU (sec)
10	10	2	1	0	1	2	20	3	0.15	0.01	14	29
10	10	4	1	0	3	1	20	5	0.03	0.03	42	3417
10	10	6	1	0	8	0	20	8	- *	-	68	25900
10	10	8	1	0	7	0	20	8	0.26	0.13	61	5784
10	10	10	1	0	7	0	20	7	- *	-	66	6137
20	20	2	1	0	1	2	20	9	0.10	0	47	58
20	20	4	1	0	8	1	20	15	0.22	0	156	1338
20	20	6	1	0	21	2	20	36	0.20	0.05	326	25900
20	20	8	1	0	25	2	20	39	0.25	0.06	344	4079
20	20	10	1	0	36	0	20	41	0.54	0.27	512	25900
30	30	2	1	0	8	3	20	76	0	0	218	151
30	30	4	1	0	22	3	20	87	0.06	0	280	438
30	30	5	1	0	67	1	2	84	0.35	0.01	720	25900
40	40	2	1	0	50	3	20	345	0.06	0	687	331
40	40	4	1	0	89	3	20	343	0.10	0	830	5201
40	40	5	1	0	78	3	20	287	0.04	0.01	1200	25900
50	50	2	1	0	42	2	20	100	0.05	0	1176	6151
50	50	4	1	0	34	3	20	428	0.01	0	942	878
50	50	5	1	0	124	3	20	807	0.01	0	1214	25900

Table 3. (TP) problems with several restrictive side constraints (* indicates that LR fails to identify a feasible solution)

			RP								LR	
			Phase I			Phase II						
n	m	k	$c\tau_1$	MR	CPU (sec)	$c\tau_2$	MR	total CPU (sec)	RD		RD (sec)	CPU (sec)
10	10	1	1	0	2	3	20	18	0.63		0.63	14
10	10	2	4	20	348	3	20	504	5.75	*	-	234
10	10	4	3	20	221	3	20	596	2.69	*	-	103
10	10	6	2	20	187	3	20	473	0.67	*	-	133
10	10	8	7	0	783	3	20	1072	0.42	*	-	104
10	10	10	10	20	1012	3	20	1200	1.83	*	-	150
20	20	1	1	0	6	3	20	78	1.17	*	-	41
20	20	2	2	20	135	3	20	314	1.95	*	-	437
20	20	4	4	20	307	3	20	566	1.79	*	-	763

Tables 1 and 2 summarize the results for problems of type (TP) with one and several side constraints, respectively. As far as CPU time is concerned, *RP* is faster, but as far as precision is concerned, the RD deviation is smaller for *LR*. Furthermore, it is easier for RP and LR to find out that a problem is infeasible (the infeasible problems are identified by a * in Table 2).

The fact that $c\tau_1 = 1$ for all problems solved in Tables 1 and 2 indicates that the additional side constraints are not very restrictive since the *recursive procedure RP* can reach the feasible domain using only one exchange at each iteration during Phase I. Hence, other problems with smaller values for the right-hand side terms b_k were generated, and the results are summarized in Table 3. On the one hand, the results indicate that *procedure LR* fails to find a feasible solution for several of these problems. On the other hand, to obtain a feasible solution, *RP* requires several forward motions in the iterations during Phase I, and the value of MR during this phase has to be increased in several cases. Hence solution time increased. These observations indicate that, whenever the additional side constraints are restrictive, it seems appropriate to use several forward motions in the iterations during Phase I of *RP* to identify a feasible solution.

Next, problems of type (APSC) are solved. The results in Table 4 indicate that the behavior of the procedures is similar to the one observed in Tables 1 and 2. However, as in Table 3, *LR* fails to find a feasible solution for some problems whereas *RP* and XMP can. This is consistent since equality constraints are, in general, more restrictive than inequality constraints. Hence these numerical results seem to indicate that a better strategy to deal with these problems would be to use the *RP* procedure with appropriate values of τ and MR to reach the feasible domain and then to proceed with the *LR* procedure to improve the objective function. Indeed, the RD value is, in general, smaller for solutions generated with *LR*.

The purpose of these numerical results is to indicate the procedures behaviour,

Table 4. (APSC) problems (* indicates that LR fails to identify a feasible solution whereas RP and XMP can; ** indicates that LR, RP and XMP fail)

			RP								LR		XMP
			Phase I			Phase II							
n	m	k'	$c\tau_1$	MR	CPU (sec)	$c\tau_2$	MR	total CPU (sec)	RD		RD	CPU (sec)	CPU (sec)
10	10	2	1	0	1	1	20	3	0.38		0.16	19	144
10	10	4	2	0	4	1	20	4	0.48	*	-	25	583
10	10	6	5	0	48	1	20	48	0.31	*	-	40	612
10	10	8	1	0	7	1	20	7	0.83	*	-	59	1058
10	10	10	1	0	10	1	20	10	-	**	-	101	25900
20	20	2	1	0	11	1	20	15	0.54		0.54	226	209
20	20	4	3	0	29	1	20	34	0.69		0.16	267	554
20	20	6	10	0	96	1	20	101	0.75		0.21	418	1475
20	20	8	2	0	46	1	20	49	0.40		0.49	484	2245
20	20	10	10	0	416	1	20	422	0.48		0.45	686	3388
30	30	2	1	0	22	1	20	30	0.54		0.36	646	252
30	30	4	2	0	37	1	20	46	0.50		0.29	1063	1490
30	30	5	1	0	71	1	20	81	0.39		0.38	1210	4483
40	40	2	1	0	30	1	20	63	0.18	*	-	1208	1540
40	40	4	5	0	213	1	20	232	0.80		0.45	1211	5317
40	40	5	2	0	123	1	20	142	0.74		0.31	1211	2080
50	50	2	2	0	204	1	20	227	0.61		0.57	1217	2756
50	50	4	1	0	171	1	20	224	0.53		0.51	1217	6015
50	50	5	1	0	411	1	20	444	0.79		0.60	1219	5993

and they are given in terms of randomly generated problems similar to those found in [18]. In fact when dealing with specific applications [1,2,5,6,9,20], very efficient computer implementations are required to deal with these large scale specific problems.

Both *LR* and *RP* procedures should be included in any decision support system since *LR* provides also a lower bound allowing an estimate of the solution quality. Finally, in the applications mentioned above, the objective functions are quite irrelevant since the users are already very satisfied to identify a feasible solution or a solution with few constraint violations.

References

[1] J. Aubin, Une approche heuristique pour la confection d'horaires de CEGEP, Master Thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué. (1985).

[2] J. Aubin and J.A. Ferland, A large scale timetabling problem, Comput. Oper. Res. 16 (1989) 67-77.

[3] M. Carter, A survey of practical applications on examination timetabling, Oper. Res. 34 (1986) 193-202.

[4] J.A. Ferland, CAT: Computer aided timetabling, Publication no. 668, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué. (1988).

[5] J.A. Ferland and C. Fleurent, Computer aided scheduling for sport league, Infor. 29 (1991) 14–25.

[6] J.A. Ferland and S. Roy, Timetabling problem for universities as assignment of activities to resources, Comput. Oper. Res. 12 (1985) 207–218.

[7] M.L. Fisher, An applications oriented guide to Lagrangean relaxation, Interfaces 15 (1985) 10–21.

[8] M.L. Fisher, R. Jaikumar and L.N. van Wassenhove, A multiplier adjustment method for the generalized assignment problem, Working paper no. 81-07-06, The Wharton School, University of Pennsylvania, Philadelphia, PA (1981).

[9] C. Fleurent, Méthodes heuristiques pour la conception de calendriers sportifs, Master Thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué. (1987).

[10] A.M. Geoffrion, Lagrangean relaxation for integer programming, Math. Programming Stud. 2 (1974) 82–114.

[11] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13 (1986) 533–549.

[12] F. Glover, The general employee scheduling problem: an integration of MS and AI, Comput. Oper. Res. 13 (1986) 563–573.

[13] F. Glover, Tabu search, CAAI Rep. 88-3, Center for Applied Artificial Intelligence, University of Colorado, Boulder, CO (1988).

[14] M. Held, P. Wolfe and H.P. Crowder, Validation of subgradient optimization, Math. Programming 6 (1974) 62–88.

[15] A. Hertz and D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (1987) 345–351.

[16] C. Laporte and S. Desroches, Examination timetabling by computer, Comput. Oper. Res. 11 (1984) 351–360.

[17] A. Lavoie, Méthode d'échanges pour problèmes d'affectation, Master Thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué. (1989).

[18] J.B. Mazzola and A.W. Neebe, Resource-constrained assignment scheduling, Oper. Res. 34 (1986) 560–572.

[19] C.T. Ross and R.M. Soland, A branch and bound algorithm for the generalized assignment problem, Math. Programming 8 (1975) 91–103.

[20] S. Roy, Problème d'affectation généralisée avec conflits: application au problème de confection d'horaires de cours, Master Thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Qué. (1982).